CALIFORNIA INSTITUTE OF TECHNOLOGY

PASADENA, CALIFORNIA

Silicon Structures Project

MEMO # 3

Two Rail Restoring Logic Blocks--
An Alternative Logic Model

by

Stephen Trimberger

June 20, 1980

Two-Rail Restoring Logic Blocks -- An Alternative Logic Model

All current functional description layout systems work as follows: a **structure** is pre-specified, a **function** is supplied by the user and the **physical** layout is generated. Thus the three parts of the design are produced. The problem with these systems is that much effort has gone into optimizing the function and the physical layout, when the real loss in density and speed is in the choice of a poor structure.

Why are people so much better than machines at laying out logic from functional descriptions? The reason is due, at least in part, to the fact that people have many layout strategies to use when implementing logic.

A much better functional description layout system could be produced if it had a choice of a number of possible structures. The first step in the design of such a system is the identification and specification of acceptable structures. In this document, I describe two new layout structures that may be better than conventional automatic layout models for a class of problems.

Although this model, or floorplan is capable of laying out any function, it is not anticipated that this will necessarily be the case. However, this structure will lay out many functions much more efficiently than current methods in terms of area, speed of execution and power dissipation.

The Current Mess

At present, in order to lay out circuits from a functional description, the designer must use polycell, gate arrays or a PLA. These are not the only structures that are capable of laying out arbitrary functions in LSI. Here I propose two more structures and an associated description language. These structures model well the way much layout is perceived in MOS technologies.

Polycell and gate arrays do not include in their design model the concept of a pass transistor. This is reasonable because pass transistors are not available in every technology. However, we have them and we want them.

PLAs on the other hand make extensive use of pass gates, but only in a ratherrigid style. Much generality is lost due to the forced regularity of the structure. That generality could be used to greatly reduce the area needed to implement simple functions.

Basic Tenets of LSI Layout

1. Power and ground are parallel to one another. Otherwise they cross, requiring crossover contacts and interconnect. Such connections for the power supply is very bad design practice, so this restriction to parallelism is a reasonable one to makefor a layout system.

2. Control flows perpendicular to data. This is a design methodology that is as old as the buildings here at Caltech. Where control and data cross there exists logic function. It is only in cases where data enters the control path or vice-versa where this is violated, but this is a rare case. We can assume with little loss of generality that control and data signals will flow perpendicular to one another.

3. Control performs function on data with pass gates. Data are combined and outputs are selected with control functions gating pass transistors. This model can be assumed with little loss of generality, but one must be careful about the definition of control signals.

4. Pass gate structures are functional blocks and are alternated with restoring logic stages. This allows fast, low-power computation to take place in pass gate structures and signals will be restored in the restoring logic section. Acceptance of this restriction requires no loss of generality of layout. If you allow null pass structures, this encompasses all cases. (The case where pass gate structures are concatenated indefinitely is of little importance because the signal will propagate too slowly or not at all.

Two Rail Layout

The first two constraints above leave us two options: power and ground parallel to data (perpendicular to control) or perpendicular to data (parallel to control).

Throughout the remainder of this document, the former will be used in examples. The latter will be mentioned only briefly, but all arguments apply to that structure as well.

The standard demo around Caltech is the two-rail shift register cell shown in figure 1 and schematized in figure 2. The shift register cell obeys all four constraints from the previous section. The cell has three interesting parts: a pullup structure (PU), a pulldown structure (PD) and a pass gate structure (PS). The rest of the circuit is highly-constrainted interconnect and can be easily automated.

Therefore a two-rail logic block (TRLB) can be fully specified by supplying the pullup, pulldown and passgate structures.

PullUp Structure

The ratio of the pullup size to the pulldown size is dependent on the input signal to the pulldown transistor. The ratio is different if the input has passed under apass gate. This ratio can be computed automatically. If the pulldown transistors are assumed to be minimum size, then the size of the pullup is settled. However, if this gate must drive a large load, the driving power might be greater. The user-defined size of the pullup gives the output driving power of the restoring logic stage and the pulldown section can be scaled appropriately.

PullDown Structure

Data elements are combined in the pulldown structure as shown in figure 3. Pulldown structures can be very large and complex. The complexity is limited because overly complex NAND structures cause the same speed problem that was noticed with pass gates.

The ratios of the height to width of the transistors in the pulldown is dependent on the pulldown series/parallel structure and the input to the gate of the transistor. The absolute sizes of the transistors are specified so as to conform to the proper ratio with the pullup. Therefore the user need only specify the structure on the pulldown and the signals on the gates of the transistors in order to absolutely

specify the mask geometry. If there are no signals specified, neither the pulldown nor the pullup are needed.

Therefore, the pulldown structure can be treated as a serial-parallel two-terminal network. One terminal is the pullup and output node, the other terminal is ground. Serial connections, NAND operations, can be represented by simply listing the input names serially. Parallel connections, NOR operations, can be represented by a vertical bar between names. Of course, parentheses can be used to group sub-structures as in the following: (A B) | C. This represents a serial connection of A and B in parallel with C as shown in figure 3. This represents the logic function NOR(NAND(A,B), C).

PassGate Structure

The passgate structure states simply which control signals are input to the pass gate section. This may be null, meaning that there are no pass gates following this pullup section. The control inputs need only be listed to fully specify the pass gate structure.

Generalization

Notice that as currently defined, each module can have only one output and one control input. If this were the case, we would soon have very dull modules. Therefore, we make the following generalizations:

Data signals pass through a cell, and can be used again in the next cell. Also, data need not be used in the pulldown calculation and may be passed through for use in later cells. In order to implement this, a list of data signals passed through the cell must be given.

Many control signals can be input and may form pass transistors on the output or on any of the data signals passing through the cell. In order to specify this added function, a list of data-control signal pairs must be given for each pass gate to be formed. The other gates may be undone using depletion-mode implant to make the transistor areas resistors, or a crossover may be inserted. The choice may be left up

the the design system.

A turn may be made by control or data signals so they may become a signalof the other type. This is easily handled by the capability to connect a data and acontrol line.

The simple two-rail layout structure now looks like figure 4. The user-specified parts are the pullup size (driving power), the pulldown structure, the passed data list, the passgate structure and the connected control-data list. This is really asmall amount of data for the function delivered and it can be easily automated.

Feedback and a Remarkable Analogy

It is necessary to feed back data in digital systems in order to build machines that are more than just combinational logic. There are two methods of feedback: bus routing and bitslice wiring (see figure 5). Bus routing requires O(n) horizontal wiring channels and O(n) vertical wiring channels. However, bitslice routing requires O(n) horizontal wiring channels, but only O(1) vertical wiring channels.

Bus routing is done as shown in figure 6 by connecting a set of data bitsto control bit lines. The control lines are then connected to another set of data lines parallel to the original lines. These lines feed back to more control lines which route the signals back down to the data signals.

Bitslice routing (figure 7) is implemented as a stack of one-bit bus routings. The space savings is achieved because the individual bits re-use the vertical wiring channels.

The use of the control and data connections bears a remarkable resemblance to wiring in gate arrays. This idea has been enhanced by the language used in this section. It is true that you can use this structure to build gate-array-like inefficient layout with wide wiring channels. However, just because the TRLB layout system can create a layout that looks like a gate-array, one cannot argue that it is as inefficient as the gate-array. The TRLB can be used to generate compact layouts and gate-arrays cannot.

Layout Optimizations

One advantage of the TRLB is that there is a nearly endless list of possible layout optimizations. The following list is not exhaustive. Many more optimizations can be made, as the physical structure of a TRLB is not overly constrained.

First, null PS or PD-PU structures can be eliminated from the layout, making the e resulting circuitry smaller. Redundant restoring logic stages can be removed if desired.

Snaked pullup and pulldown transistors can be inserted when such would be more area-efficient. This would allow reasonably compact drivers to be built. The decision on when to snake a transistor can be made from a global optimization strategy, such as minimize the height of the chip.

Long chains of TRLBs can be snaked to improve the aspect ratio of the resulting logic. Unused pieces of wiring can be chopped off and wiring channels re-used. All the wiring heuristics can be used to minimize wiring channels where they are used. Adjacent TRLBs can be mirrored to share one power or ground rail.

The pass gate structure can be implemented optionally as a function block or astrue crossovers.

Synchronous Pipelines

One particularly interesting special case of the TRLB is the synchronous pipeline (see figure 8). If we limit the control signals to clocks, then each TRLB isone stage in a pipeline. The stages can be simply concatenated to produce a functioning pipeline. Conglomerations of pipe stages are pipeline processors. Pipeline processors can be concatenated to make larger processors, up to the limit of clock skew, which won't be bad for a couple years yet.

Power and Ground Perpendicular to Data

Figure 9 shows a full-blown TRLB with power parallel to control. It has thesame regions as figure 4 and the correspondence between them should be obvious.

A Language for Expressing TRLBs

The lowest level language that can be used for TRLBs is one which describes each of the pulldown, passgate and bypass structures. These can be done with a simple list, as described above. An example of the language applied to a simple shift register follows:

```
F = A'          (pullup and pulldowns)
OUT = F & CLK   (passgate structure)
(anything else just passes through)
```

The following more difficult example is a very simple signal processing application, where for each output, F, F_i = NAND(X_j for j = i-10 to i). This example is shown graphically in figure 10 and figure 11.

```
DEFINE INVERT;
    F = A';     (everything else defaults)
    BYPASS B;

DEFINE NAND;
    F = (A & B)'
    OUT = (A, B) & CLK

DEFINE SIGNAL_PROCESSOR_STAGE;
    NAND, INV;   (serial composition)

DEFINE SIGNAL_PROCESSOR;
    SIGNAL_PROCESSOR_STAGE*10;     (array construction)
```

Notice that no extra specification was given to tell the INV cell to route the B signal below the GND line for BYPASS. This is an easy assumption that the layout generation program can make. There are several other low-level operations that are necessary and obvious and need not be mentioned here.

Restriction of the problem to serial pipeline processors allows a number of simplifying assumptions in the language used to discuss the layout. Such assumptions are shown above in the composition cells, where serial compositions of

cells and arrays are implied to be along the direction of the pipe.

Such simplifying assumptions can also be used to allow an easy way to express serial-parallel networks of cells, giving rise to a functional parallel language for expressing pipeline processors.

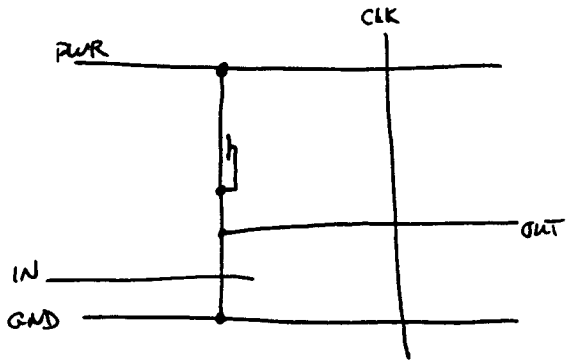

Figure 1. TRLB Shift Register Cell

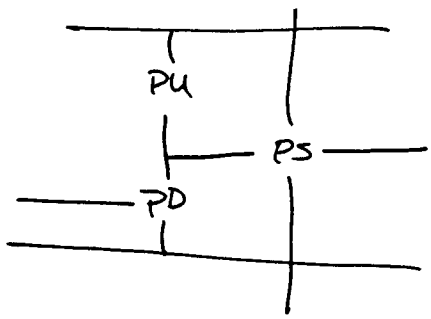

Figure 2. TRLB Schematic

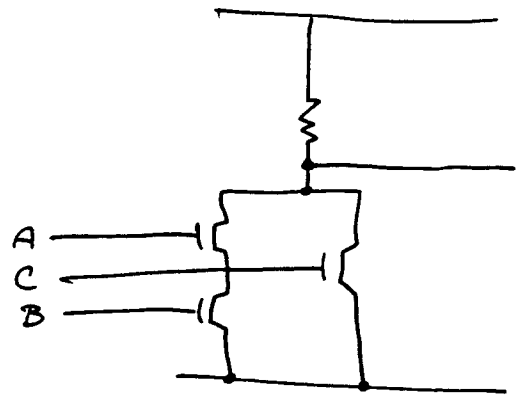

Figure 3 Pulldown Structure $\overline{(\overline{A \cdot B}) + C}$

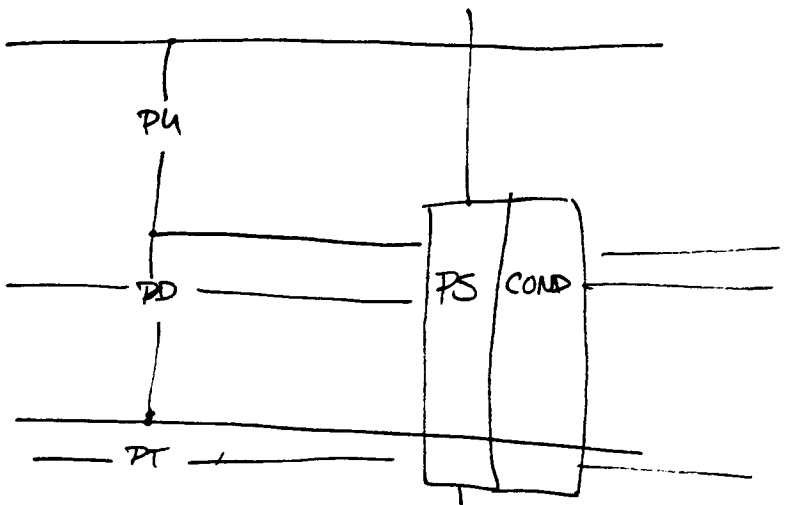

Figure 4 Full-Blown TRLB Schematic

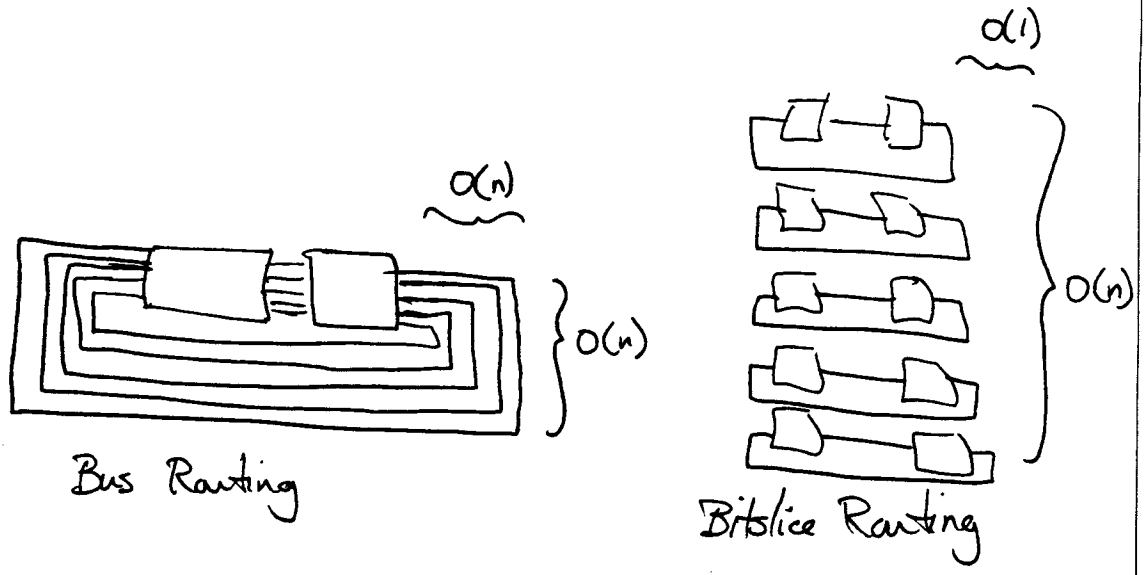

Figure 5. Routing Styles

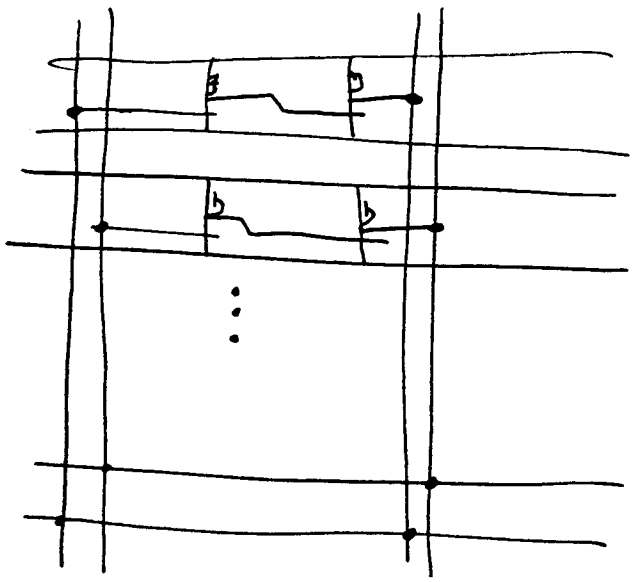

Figure 6 Bus Routing in TRLB

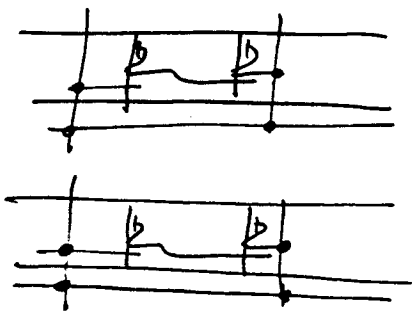

Figure 7 Bitslice Routing in TRLB

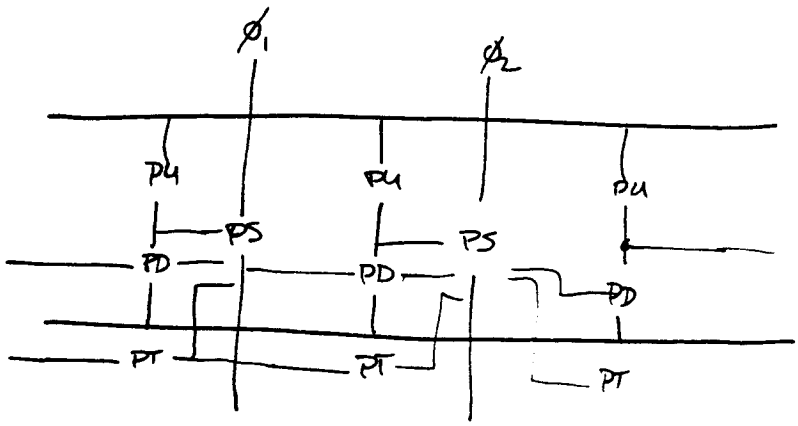

Figure 8 Synchronous Pipelines

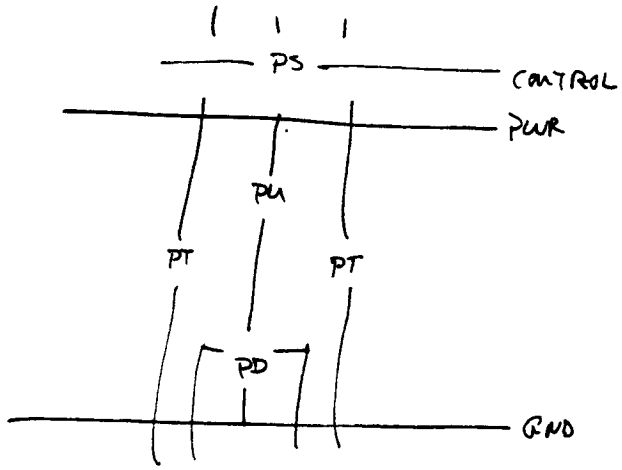

Figure 9 TRLB with power parallel to control

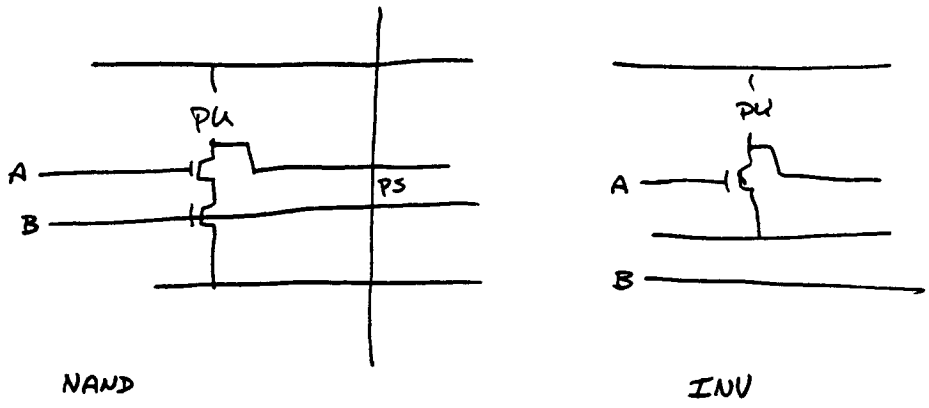

Figure 10 NAND Pipe Leaf Cells

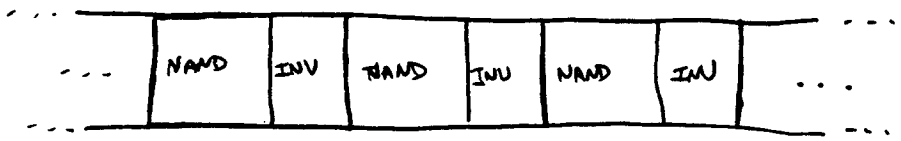

Figure 11. NAND Signal Processor